Original Report

PLACE: An Open-Source Python Package for Laboratory Automation, Control, and Experimentation

Journal of Laboratory Automation
2015, Vol. 20(1) 10–16

DOI: 10.1177/2211068214553022
jala.sagepub.com
SAGE

Jami L. Johnson[1], Henrik tom Wörden[2], and Kasper van Wijk[1]

Abstract

In modern laboratories, software can drive the full experimental process from data acquisition to storage, processing, and analysis. The automation of laboratory data acquisition is an important consideration for every laboratory. When implementing a laboratory automation scheme, important parameters include its reliability, time to implement, adaptability, and compatibility with software used at other stages of experimentation. In this article, we present an open-source, flexible, and extensible Python package for Laboratory Automation, Control, and Experimentation (PLACE). The package uses modular organization and clear design principles; therefore, it can be easily customized or expanded to meet the needs of diverse laboratories. We discuss the organization of PLACE, data-handling considerations, and then present an example using PLACE for laser-ultrasound experiments. Finally, we demonstrate the seamless transition to post-processing and analysis with Python through the development of an analysis module for data produced by PLACE automation.

Keywords

laboratory automation, Python, open source, laser ultrasound

Introduction

The open-source software (OSS) movement traces back to the 1950s,[1] when voluntary organizations such as Share—an intermediary between IBM and its users—pursued collaboration and reduced programming costs by leveraging diverse expertise and practices of academic exchange.[2] However, notable developments, such as the launch of the GNU Project in 1984[3] and Netscape's decision to release the source code for its Internet browser in 1998,[1] significantly increased interest in the movement. More recently, examples such as the Linux operating system,[4] WordPress for blogging,[5] and Google Chrome for Internet browsing[6] have further demonstrated the potential for successful open-source projects. A majority of code developed under the umbrella of "open source" is distributed either free or for a nominal charge, but the primary distinguishing feature is the freedom to use, modify, and distribute source code under a public, unrestricted license.[1,7]

Laboratories built on the principles of the open-source movement have been in existence for decades.[8] Beyond the obvious benefit of reducing laboratory running costs, there are many advantages of an open-source laboratory. Collaborative development fuels software capabilities to be expanded beyond what is plausible through the efforts of a single development team.[9] When open-source code is effectively disseminated, there is a community of programmers available for testing.[10] Therefore, some of the burden of validation is taken over by the community, errors are reduced, and reliability and traceability are improved.[9] In the spirit of Eric Raymond's proclaimed Linus's Law, "given enough eyeballs, all bugs are shallow."[11]

Perspectives toward OSS are changing, as the tools we use become increasingly open source. Still, laboratory automation remains an uncommon field of open software development.[9] When proprietary automation packages are used, specialized software is often required to both acquire and read the data. This causes difficulties in exchanging data across computers or operating systems and incurs extra cost and licensing issues.[12] For collaborative research, it is beneficial for all contributors to use the same software. Open-source systems facilitate this with a minimal amount of cost and inconvenience. Altogether, it is favorable to use a software platform that is completely open source, from the

[1]Department of Physics, University of Auckland, Auckland, New Zealand
[2]Institute for Nonlinear Dynamics, Georg-August-Universität, Göttingen, Germany

Received Jun 25, 2014.

Corresponding Author:
Jami L. Johnson, Department of Physics, University of Auckland, 38 Princes Street, The Science Center, Auckland 1010, New Zealand.
Email: jami.johnson@auckland.ac.nz

Table 1. Hardware Controlled by PLACE Automation.

Company	Instrument	Description
Polytec (Irvine, CA)	OFV-5000 Controller, OFV-505 Sensor Head	Laser Doppler vibrometer
Spectra-Physics (Newport, Irvine, CA)	Quanta-Ray INDI	Pulsed Nd:YAG laser
Newport	XPS Motion Controller	Driver for Newport stages
Newport	M-IMS1000LM	Linear motor stage
Newport	URS1000BCC	Rotation motor stage
Stanford Research Systems (Sunnyvale, CA)	DS345	Function generator
Alazar Tech (Point-Claire, Québec, Canada)	ATS660 and ATS9440	PCI oscilloscope card
Tektronix (Beaverton, OR)	TDS3014B	Oscilloscope

The Alazar Tech oscilloscope cards require a proprietary library, in addition to the PLACE driver module. Devices requiring exclusive libraries or software are often inevitable; therefore, robust design incorporating these instruments with an overall open-source system is important.

acquisition software and data storage through the data analysis and postprocessing.

Here we present PLACE, an open-source Python package for Laboratory Automation, Control, and Experimentation.[13] The package is available at https://www.github.com/johjam/PLACE and contains a core set of instrument driver modules for hardware used in the Physical Acoustics Laboratory (**Table 1**). The software is currently used to automate laser-generated ultrasound experiments, including laser ultrasonics of heterogeneous materials,[14] biomedical photoacoustic imaging,[15] and fracture characterization.[16] However, PLACE accommodates a wide range of applications and laboratory schemes, as unique modules are written for each instrument.

Python-based automation packages are in development, such as PyDAQmx,[17] pyVISA,[18] and Lantz,[19] which are designed to facilitate communication between a PC and laboratory instruments with Python. PLACE's approach differs in that we extend our open-source approach to encourage compatibility across all aspects of experimentation. In addition to instrument drivers, a collection of visualization functions and processing software has been developed for data produced by PLACE automation. These functions can be called with a single line of code and provide interactive plotting and data-processing capabilities. PLACE drivers use communication protocols standard to most personal computers, further supporting portable and flexible use. The PLACE package is extensible, such that new instruments, packages, or protocols can be easily incorporated. For example, parallel processes can be incorporated by implementing a package such as Parallel Python.[20] Devices requiring proprietary software or libraries often cannot be avoided. However, whenever possible, we aim to develop open-source drivers that stand alone, rather than as wrappers for proprietary code. Naturally, this also requires careful hardware consideration. It is our goal to expand this repository through continued internal development and contributions from the broader community.

PLACE for Laboratory Automation

Python is a powerful, mature tool that possesses many advantageous qualities for laboratory automation. It is open source, freely available, and continually growing through the contribution of an international community of developers. Guido van Rossum, the founder of Python, was motivated by the theory that "readability counts"[21]; thus, Python syntax was designed with clarity at the forefront.[22] The standard library is well debugged and equipped with extensive built-in modules. Python is flexible and extensible, competes among the fastest languages for computation speed, and requires minimal lines of code for high productivity.[22] The cost for numerical computing of large data sets has been shown to be less than 25% higher than pure Fortran code.[23] Furthermore, automation packages written in Python are virtually platform independent, as most operating systems—including Linux platforms, Windows, and Mac OS X—are supported.

The organization of complex code can be streamlined using Python's object-oriented approach. As illustrated in **Figure 1**, PLACE automation is organized with a unique driver module for each instrument. These modules contain the fundamental code for communication and control with a given instrument. Further hierarchical organization is accomplished with the use of classes, subclasses, and functions. A master script imports the driver modules and coordinates the activity of all instruments to perform the tasks required for a given protocol. This object-oriented approach seamlessly incorporates new instruments or methodologies as a laboratory grows and advances.

The PLACE modules developed inhouse rely on well-established modules in Python repositories (**Table 2**). These include NumPy for scientific computing[24] and matplotlib for plotting.[25] Data are transmitted between the acquisition PC and instruments via RS-232, ethernet, and USB using pySerial.[26] ObsPy is used to define and save header information (metadata), store data in a list-like object known as a stream,[27] and append data to a file dynamically. Saving data in an ObsPy-compatible format during acquisition

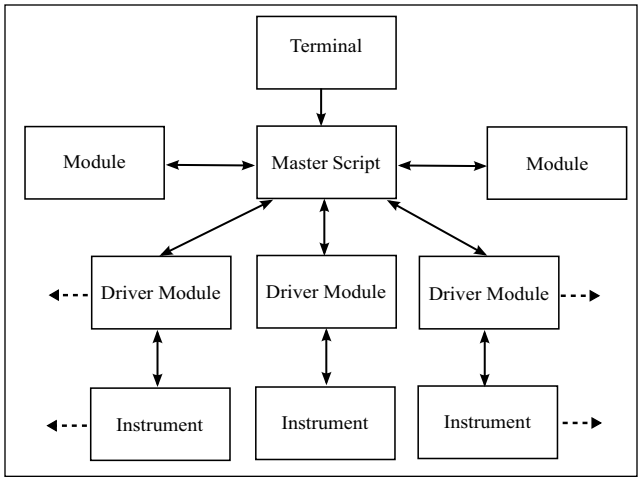

Figure 1. Basic organization of PLACE scripts and modules. Experimental protocols are defined in the terminal. The master script communicates with driver modules developed inhouse to automate laboratory instruments. Python modules available from repositories or built in to Python are used to complete tasks such as plotting, saving data, and serial communication.

Table 2. Python Packages and Versions Used by PLACE.

Package	Version
Python	2.7.7
Obspy	0.9.0
pySerial	2.7
NumPy	1.8.1
matplotlib	1.3.1
SciPy	0.14.0
h5py	2.3.0
obspyh5	0.1.0

allows us to make full use of ObsPy's capabilities for post-processing and analysis. ObsPy is designed for processing seismological data but has powerful functionality for processing waveforms or time series of all kinds. The package continues to develop at a rapid pace, yet ObsPy's test-driven approach[28] has yielded stable releases on a regular basis.

Data Handling

Robust data handling is essential for automated laboratory processes to ensure integrity, accountability, and preservation of data.[28] We ensure effective data management by incorporating detailed header information, using an efficient and well-established data format, and dynamically saving data to file as it is acquired.

A custom header format was created, which saves the unique parameters for each trace (A-scan) as it is acquired. We chose to save metadata in a header during acquisition, as opposed to using a database, with goals of convenience and robust coupling of the metadata to the data it describes. Headers standardize laboratory records, simplify postprocessing, and essentially serve as an electronic laboratory notebook. As an added benefit, both PLACE and ObsPy functions automatically read and use header parameters, minimizing the need to enter arguments manually. For example, a voltage is acquired from our vibrometer receiver. The calibration factor required to convert the voltage to physical units, such as displacement or particle velocity, is dependent on the choice of decoder and settings of the vibrometer receiver. These values are held by the vibrometer hardware, which is queried during acquisition and recorded in the header. The ObsPy `read` function automatically applies this calibration factor upon import. All ObsPy processing that is applied to a stream is also recorded in the header, which further supports accurate record keeping and reproducibility. Example header information for a single trace is shown below.

```
   network :
   station :
   location:
   channel : CHANNEL_B
  starttime: 2014-04-30T21:51:25.449480Z
    endtime: 2014-04-30T21:51:25.449890Z
  sampling_rate: 10000000.0
     delta : 1e-07
       npts: 4096
      calib: 50.0
   _format : H5
   averages: 200
calib_unit: nm/V

    comments: shale sample with brass jacket
    decoder: DD-300
  decoder_range: 50mm/s/V
      focus : 0.919596354167
  max_frequency: 20000000.0
   position: 180.0
 position_unit: deg
processing: ["filter:highpass:{'freq':
100000.0}"]
  scan_time: 363.84270525
   source_energy: 200 mJ
time_delay: 0.0
```

Data are saved in HDF5, the free and open standard hierarchical data format. HDF5 is designed for large quantities of complex data and efficient I/O. In addition, HDF5 is portable across operating systems[29] and supported by most programming languages, including C++, java, MATLAB, and Scilab.

Traces with headers are saved dynamically to an HDF5 file through the ObsPy, H5py,[30] and obspyh5[31] packages. Within the file, traces are appended to a stream. Saving continuously reduces concerns with limited buffer memory and ensures data

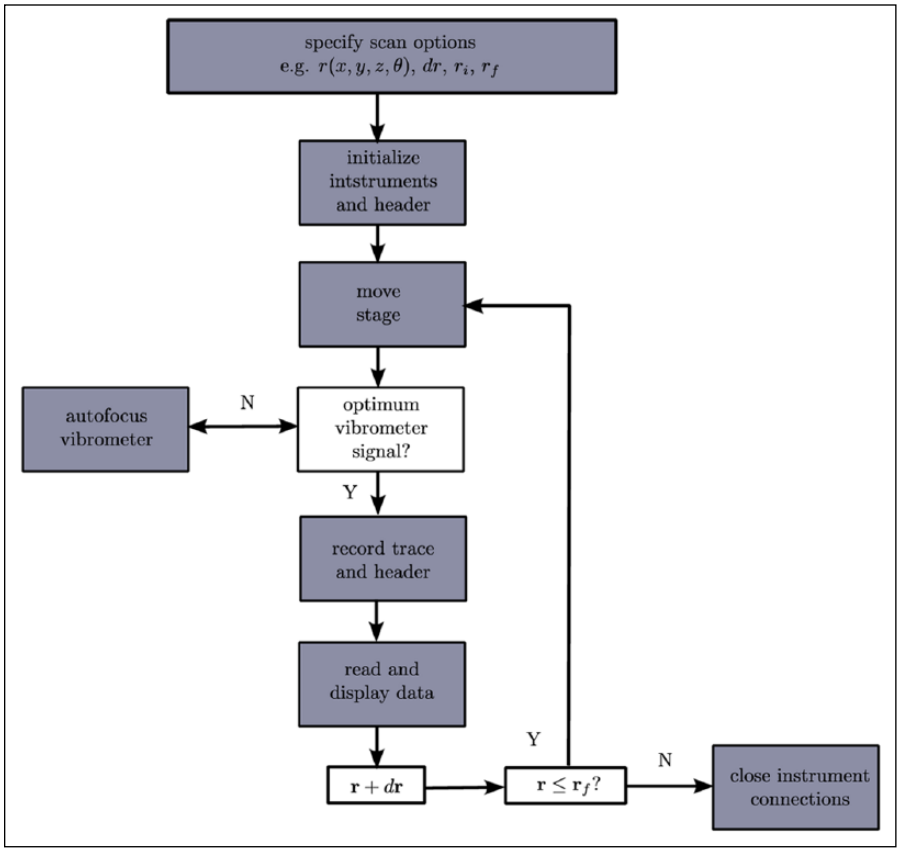

Figure 2. Example implementation of PLACE automation for a laser-ultrasound scan, where **r** is the initial position vector for each stage, d**r** is the increment for each dimension, and $\mathbf{r}_f$ is the final position vector.

are preserved if acquisition is disturbed. If an experiment must be interrupted, automation can be paused and restarted from the last position without creating a new data file. In addition, multiple experiments with different protocols can be saved to a single file. The unique header for each trace ensures the discernibility of the experiments. Furthermore, when only the metadata are required, Obspy can efficiently load the trace headers, without unpacking the waveform data.

Implementation of PLACE Automation

An example using PLACE for a laser-ultrasonics experiment is presented in **Figure 2**. First, all options are defined in the terminal and imported to the master script. Subsequently, the instruments are initialized and header information is defined. To perform specific tasks, the master script interfaces with individual modules. For example, prior to recording a trace, the master script reads the signal strength of the vibrometer sensor head through the oscilloscope card module. If the signal is suboptimal, the vibrometer module is employed to focus the sensor head until a satisfactory signal level is reached. Subsequently, the trace is recorded with the oscilloscope card module. The frequency content of laser-generated ultrasound experiments ranges across the ultrasound spectrum, with up to tens of megahertz generated in applications such as biomedical photoacoustic imaging. We are limited strictly by the receiver hardware, as acquisition with PLACE and the oscilloscope card allow frequencies up to 125 MHz to be recorded. Once acquired, the trace and header are saved to an HDF5 file and displayed.

Throughout an experiment, progress can be monitored both via the terminal output and graphically (**Figure 3**). After each trace is acquired, the most recent trace and the cumulative wavefield are displayed with matplotlib. The master script prints updates to the terminal with each trace, including the approximate time remaining, current trace number and stage position, and notifications when the vibrometer is refocused. The experiment continues until conditions for completion are met.

The setup for a rotational scan of a shale cylinder with P-wave anisotropy is shown in **Figure 4**. The source and receiver were aligned and stationary. The shale was mounted on a 360° continuous rotational stage, and waveforms were acquired over 180°. All experimental parameters were defined in the terminal, and the master script was executed using the following line of code:

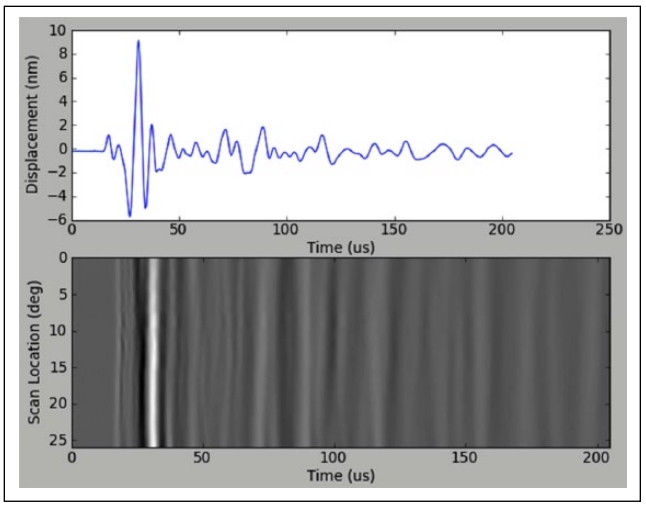

Figure 3. Screenshot of the most recent trace and cumulative wavefield during laser-ultrasound scan of a sandstone sample.

```
python Scan.py -t 256 -c B -a 200 -s rot
-i 0 -x 1 -f 180 -n shale_brass -d DD-300
-e 200
--comments='shale sample with brass
jacket'
```

where each parameter is defined in the example Scan.py file provided with the PLACE package. The resulting waveform data are shown in **Figure 4**.

PLACE Data Analysis

The first release of PLACE includes an analysis module, PALplots. Several visualization functions are incorporated: a contour plot for viewing the raw waveforms (**Fig. 4**), a "wiggle" plot that highlights the location of dominant events, and a function for viewing the frequency-wavenumber (f-k) spectrum. In addition, an f-k filter is included for suppressing unwanted or interfering waves within a range of apparent velocities. A wiggle plot and f-k spectrum for the shale data are shown in **Figure 5**.

Each PALplots function provides the user with the ability to interactively select points in the figure. For the `contour` and `wiggle` plots, position and time coordinates can be dynamically selected and saved to a file. This is useful for recording event locations and the time of arrival of particular waves. Any line through the origin and a point in the f-k domain has a slope corresponding to a velocity value (velocity = frequency/wavenumber). The `fkfilter` function allows the user to select a pair of coordinates dynamically and annotates the figure with the corresponding velocities. When submitted, these coordinates define the range of velocities in the data that will be suppressed by the filter. Waves that arrive with an apparent velocity within this range will be removed.

Conclusions and Future Work

PLACE is an open source, freely available Python package for laboratory automation and analysis. The clean, object-oriented design provides the freedom to expand and customize PLACE to meet the needs of diverse laboratories and experiments. Future work includes the development of driver modules for scanning mirrors, amplifiers, and additional stages for three-dimensional scanning. We plan to expand the PLACE library of processing software for laser-generated ultrasound experiments, as well as integrate code for resonant ultrasound spectroscopy. Our open-source approach spans across the spectrum of experimentation and has also fueled the development of open-source hardware. This includes projects such as seismometers for primary and secondary education[32] and an open-source laser Doppler vibrometer. PLACE has been thoroughly tested under the Linux operating system and has shown functionality across platforms. However, future work includes additional testing under Windows and Mac OS X operating systems. We aim

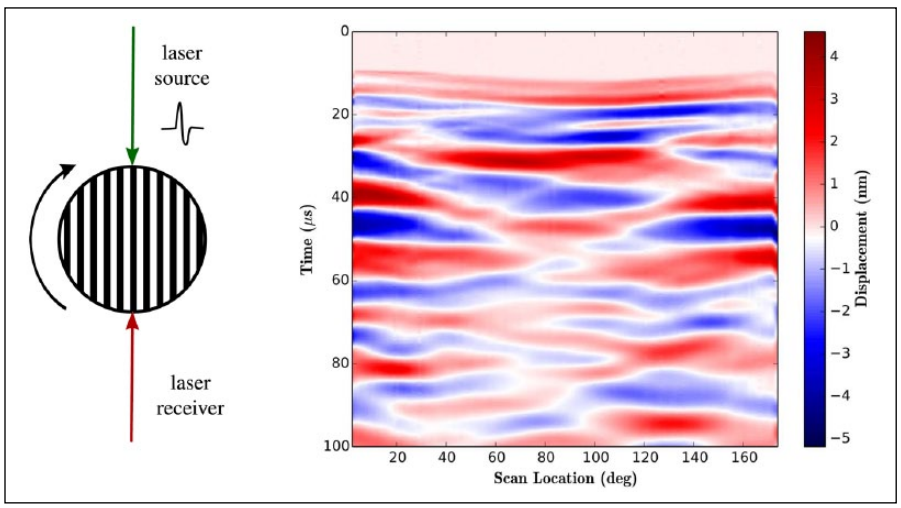

Figure 4. (**Left**) Setup for rotation scan of a shale sample with brass jacket. (**Right**) Resulting laser-ultrasound wavefield of shale. The colored regions represent the arrival of ultrasound waves after propagating through the sample. The shale shows significant P-wave anisotropy: the speed of sound in the rock varies depending on the direction of propagation. Note the delayed arrival time of waves around 90° compared with those at the margins of the scan.

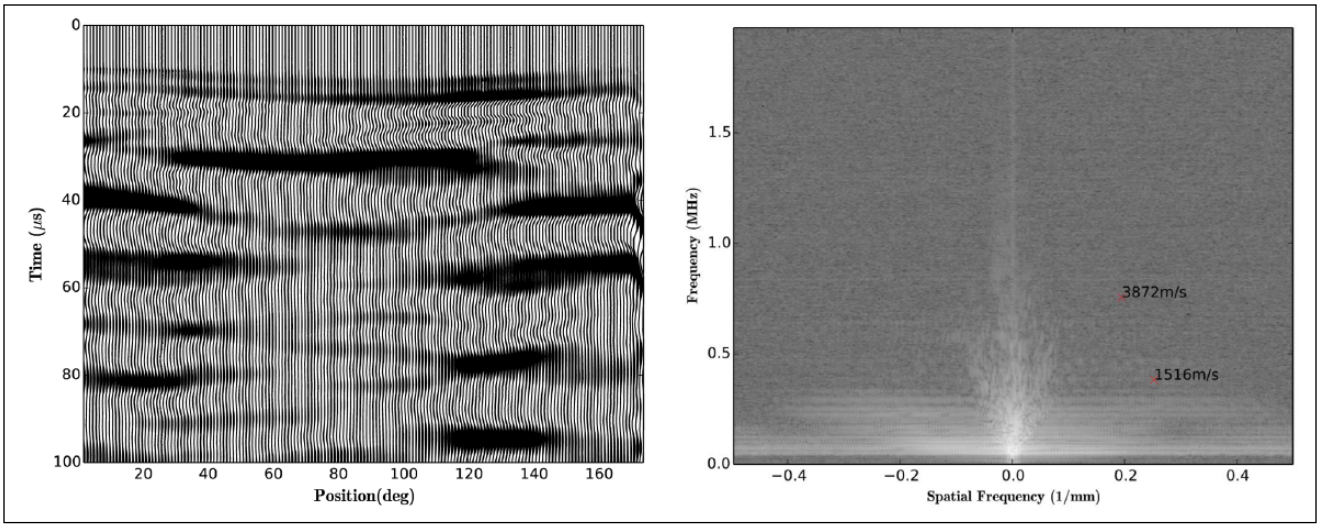

Figure 5. Example plots for the shale sample with brass jacket created with the PALplots module. (**Left**) Wiggle plot. (**Right**) Frequency-wavenumber spectrum of shale. Selected points are shown with the corresponding apparent velocity annotated. These points can be used to define the limits of an f-k filter.

to actively develop additional modules and software and we encourage contribution from the community.

Acknowledgments

J.L.J. thanks the University of Auckland Doctoral Scholarship for doctoral study support. The authors would like to give a special thanks to Dr. John Scales, Colorado School of Mines, for input that has improved this article and PLACE, as well as the naming of the package. In addition, thanks to Dr. Thomas Lecocq, the Royal Observatory of Belgium, for his helpful suggestions.

Declaration of Conflicting Interests

The authors declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The authors disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: J.L.J. thanks the University of Auckland Doctoral Scholarship for doctoral study support.

References

1. Androutsellis-Theotokis, S; Spinellis, D.; Kechagia, M.; et al. Open Source Software: A Survey from 10,000 Feet. *Found. Trends. Tech. Info. OM*. **2011**, *4*, 187–347.
2. Akera, A. Voluntarism and the Fruits of Collaboration: The IBM User Group, Share. *Technol. Cult*. **2001**, *42*, 710–736.
3. Stallman, R. M. The GNU Manifesto. *Dr. Dobb's J*. **1985**, *10*, 30–35.
4. Torvalds, L. The Linux Edge. *Commun. ACM*. **1999**, *42*, 38–39.
5. Douglass, R. T; Little, M.; Smith, J. W. *Building Online Communities with Drupal, phpBB, and WordPress*; Springer: New York, 2006.
6. Gray, J. Google Chrome: The Making of a Cross-Platform Browser. *Linux J*. **2009**, *185*.
7. Stallman, R. M. The GNU Operating System and the Free Software Movement. In *Open Sources: Voices from the Open Source Revolution*; DiBona, C., Ockman, S., Stone, M., Eds.; O'Reilly Media: Sebastopol, CA, 1999; pp 53–70.
8. Scales, J. A; Smith, M. L; Ballüder, K. Selecting an Operating System, Part III: Unix in a Laboratory Environment. *Comput. Phys*. **1995**, *9*, 584–588.
9. Benn, N. D; Liscouski, J. Discussion of Open-Source Methodologies in Laboratory Automation. *J. Lab. Autom*. **2009**, *14*, 82–89.
10. Morgan, L.; Finnegan, P. Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms. In *Open Source Development, Adoption and Innovation*; Feller, J., Fitzgerald, B., Scacchi, W.; et al., Eds.; Springer: New York, 2007; pp 307–312.
11. Raymond, E. The Cathedral and the Bazaar. *Knowledge, Technology & Policy* **1999**, *12*, 23–49.
12. Nielsen, K.; Andersen, T.; Jensen, R.; et al. An Open-Source Data Storage and Visualization Back End for Experimental Data. *J. Lab. Autom*. **2014**, *19*, 183–190.
13. Van Rossum, G.; Drake, F. L., Jr. *Python Tutorial*; Centrum voor Wiskunde en Informatica: The Netherlands, 1995.
14. Blum, T. E.; Adam, L.; van Wijk, K. Noncontacting Benchtop Measurements of the Elastic Properties of Shales. *Geophysics* **2013**, *78*, C25–C31.
15. Johnson, J. L.; van Wijk, K.; Sabick, M. Characterizing Phantom Arteries with Multi-Channel Laser Ultrasonics and Photo-Acoustics. *Ultrasound Med. Biol*. **2014**, *4*, 513–520.
16. Blum, T. E.; van Wijk, K.; Snieder, R. Scattering Amplitude of a Single Fracture under Uniaxial Stress. *Geophys. J. Int*. **2014**, *197*, 875–881.
17. Cladé, P. PyDAQmx: A Python Interface to the National Instruments DAQmx Driver. http://packages.python.org/PyDAQmx (accessed Aug 21, 2014).

18. Bronger, T.; Thalhammer, G. PyVISA: Python Wrapper for the VISA Library. http://pyvisa.readthedocs.org (accessed Aug 21, 2014).
19. Grecco, H. E.; Masip, M.; Jais, P.; et al. Lantz: An Automation and Instrumentation Toolkit in Python. http://lantz.glugcen.dc.uba.ar (accessed Aug 21, 2014).
20. Palach, J. *Parallel Programming with Python;* Packt Publishing Ltd: Birmingham, UK, 2014.
21. Day, C. Python Power. *Comput. Sci. Eng*. **2014**, *16*, 88–88.
22. Prechelt, L. Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. *Adv. Comput*. **2003**, *57*, 205–270.
23. Yuffa, A. J.; Scales, J. A. Object-Oriented Electrodynamic S-matrix Code with Modern Applications. *J. Comput. Phys*. **2012**, *23*, 4823–4835.
24. Oliphant, T. E. Python for Scientific Computing. *Comput. Sci. Eng*. **2007**, *9*, 10–20.
25. Hunter, J. D. Matplotlib: A 2D Graphics Environment. *Comput. Sci. Eng*. **2007**, *9*, 90–95.
26. Liechti, C. pySerial. http://pyserial.sourceforge.net (accessed May 8, 2014).
27. Beyreuther, M.; Barsch, R.; Kirscher, L.; et al. ObsPy: A Python Toolbox for Seismology. *Seismol. Res. Lett*. **2010**, *81*, 530–533.
28. Iverson, M.; Frankel, M. S.; Siang, S. Scientific Societies and Research Integrity: What Are They Doing and How Well Are They Doing It? *Sci. Eng. Ethics* **2003**, *9*, 141–158.
29. Koziol, Q. HDF5. In *Encyclopedia of Parallel Computing*; Padua, D., Ed. Springer: New York, 2011; 4, 827–833.
31. Collette, A. HDF5 for Python. http://docs.h5py.org (accessed May 8, 2014).
31. Richter, T. obspyh5. https://github.com/trichter/obspyh5 (accessed May 8, 2014).
32. van Wijk, K.; Channel, T.; Viskupic, K.; et al. Teaching Geophysics with a Vertical-Component Seismometer. *Physics Teacher* **2013**, *51*, 552–554.